Test Informatique N°1

Problème

Dans ce problème, on se propose avoir créé le module `file.py` qui implémente la structure file où on a défini les fonctions suivantes :

- `creer_file()` permettant de créer et retourner une file vide.
- `enfiler(f,e)` permettant d'ajouter un élément `e` à la file `f`.
- `defiler(f)` permettant de défiler et retourner la tête de la file `f`.
- `est_vide(f)` à résultat booléen, permettant de vérifier si la file `f` est vide.

Le système d'exploitation d'un ordinateur doit gérer l'allocation du processeur aux différents processus (tache /application à exécuter par le processeur) à exécuter. Parmi les algorithmes permettant l'ordonnancement de l'allocation du processus, on applique le *Round-Robin* dont le principe est le suivant :

— Les processus en attente de l'exécution seront stockés dans une file d'attente : une structure FIFO « premier entrant premier sortant »

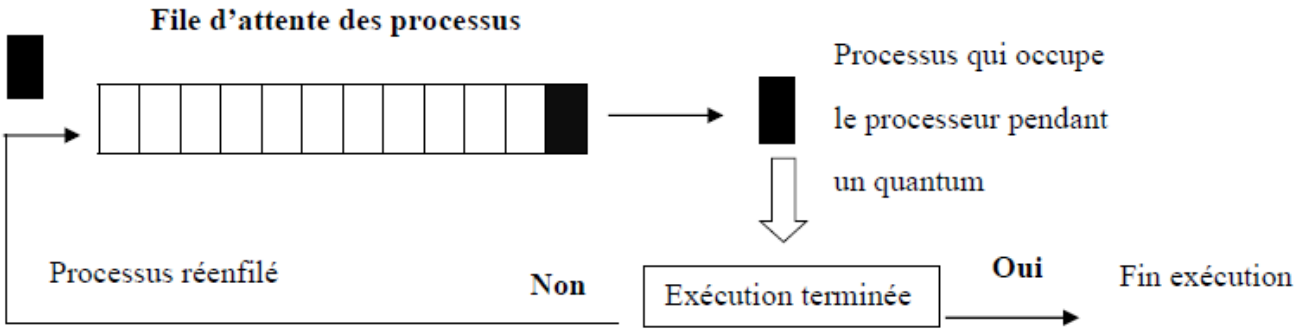
— A son tour, le processus occupe le processeur mais pendant un temps fini appelé *quantum* (unités de temps) même si ce quantum n'est pas suffisant pour qu'il termine son exécution.

— Si le processus n'a pas terminé son exécution, il sera réenfilé dans la file.

— Tous les processus disposent du même quantum de temps que les autres.

— Chaque processus peut passer par trois états :

- *Prêt* : en attente d'exécution pour la première fois.
- *Actif* : en cours d'exécution (c'est-à-dire défilé de la file).
- *Interrompu* : réenfilé dans la file.



Dans la suite, chaque processus sera décrit par une liste de la forme :

$$[Identifiant : Temps\_traitement, Temps\_Attente, T]$$

avec :

– Identifiant : une chaine de caractère qui représente l'identifiant du processus d'une façon unique

– Temps_traitement : Un entier qui représente le temps d'exécution du processus en nombre de quantums

– Temps_Attente : Entier qui représente le temps que le processus doit attendre avant de passer au processeur (nb*quantum)

– T est un tuple, de la forme (prêt, actif, interrompu) , décrivant l'état du processus comme suit :

- T= (1,0,0) si le processus est en attente (prêt).
- T= (0,1,0) si le processus est actif (défilé de la file).
- T= (0, 0,1) si le processus est interrompu.

Exemple : `P=['processus1', 5,20,(1,0,0)]`

Travail Demandé

En utilisant seulement les primitives définies dans le module `file.py` décrit en dessus, définir les fonctions suivantes :

1. Écrire une fonction `miseAjour(p, k,q)` permettant de mettre à jour le temps d'attente du processus `p` en lui affectant `k*q` avec `k` un entier représentant le nombre des quantums `q` ajoutés.
2. Écrire une fonction `affiche_etat(p)` permettant d'afficher l'état d'un processus `p` passé comme paramètre selon ce modèle : « le Processus1 est en attente ».
3. Écrire une fonction `modifier_etat(p,etat)` qui prend comme paramètre le processus `p` et un caractère `c` , représentant le nouveau état du processus : *'P'* pour *prêt* , *'A'* pour *actif* et *'I'* pour *interrompu* .
   Exemple : `modifier_etat(p,'A')` modifie l'état du processus `p` en le rendant *actif*.
4. `copyF(f)` permettant de retourner une copie `fc` d'une file `f`.
5. `taille(f)` permettant de retourner le nombre des éléments de la file `f`.
   NB : il est interdit d'utiliser la fonction prédéfinie « len »
6. `appartient_file(f,ident)` à résultat booléen , permettant de vérifier si un identifiant `ident` est un des identifiants des processus de la file `f`.
7. `ajouter(f,q)` prenant comme paramètre la file `f` ainsi que la valeur du quantum `q` et permettant de saisir et enfiler un processus dans la file `f`.
8. `reduire(f,q)` prenant comme paramètre la file `f` ainsi que la valeur du quantum `q` et permettant de réduire le temps d'attente de tous les processus d'un quantum.
9. `exécuter(f,q)` permettant d'exécuter le processus en tête de la file suivant le principe décrit en dessus.
10. `affiche(f)` permettant d'afficher tous les processus de la file `f` ainsi que leurs états.